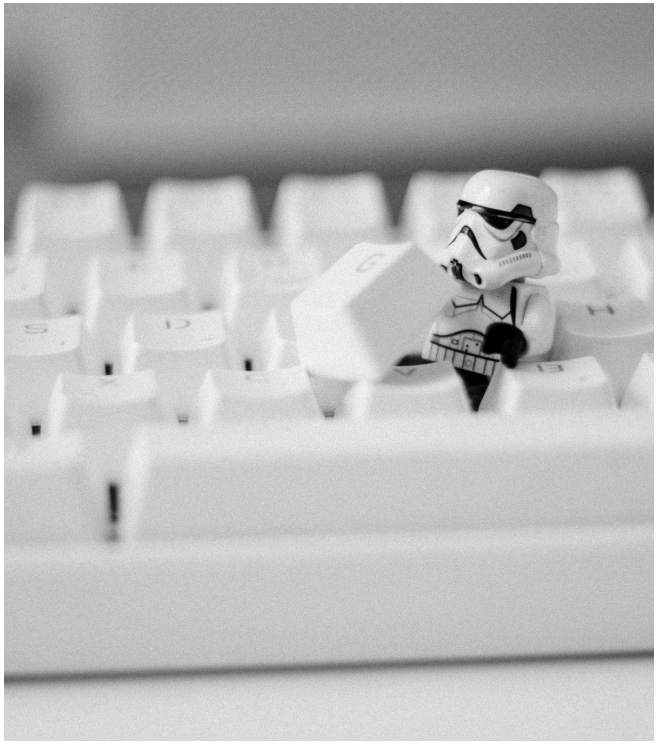
Les pratiques tech qui peuvent aider votre équipe

Jordan Chapuy



QUI QUE C'EST QUE MOI ?

17.21.9 17.21.5 3.5.19.20 17.21.5 13.15.9

JORDAN CHAPUY

Développeur iOS

Artisan du code

Blog - **https://jordanchapuy.com**

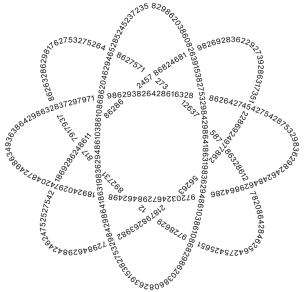
Twitter - **@Yupjoo**

(Dire que j’aime beaucoup Kaamelott, c’est pas faux)

INTENTIONS

9.14.20.5.14.20.9.15.14.19

- Découvrir des pratiques tech reconnues
- Savoir reconnaître certains problèmes
- Comprendre comment ces pratiques peuvent aider
- Accessible pour tous, à creuser par l’équipe



COLLABORER SUR LE BESOIN

3.15.12.12.1.2.15.18.5.18 19.21.18 12.5 2.5.19.15.9.14

“Les refinements sont chiants.”

“On se rend compte pendant le sprint que les besoins ne sont pas clairs. Les développeurs demandent au PO des spécifications hyper détaillées avant de commencer.”

“Le PO change d’avis, on fait ce qu'il y a écrit dans la story mais il dit que ce n'était pas ça qu’il voulait.”

EXAMPLE MAPPING

5.24.1.13.16.12.5 13.1.16.16.9.14.7

- Créer des exemples précis pour chaque règle d’une user story
- Co-création avec l’implication de tous (Devs, POs, QAs)
- Implicite rendu explicite
- Découpage plus fin

COLLABORER SUR LE CODE

3.15.12.12.1.2.15.18.5.18 19.21.18 12.5 3.15.4.5

“J’entends parler de branches, merge, commit, conflits, … Je suis perdu.”

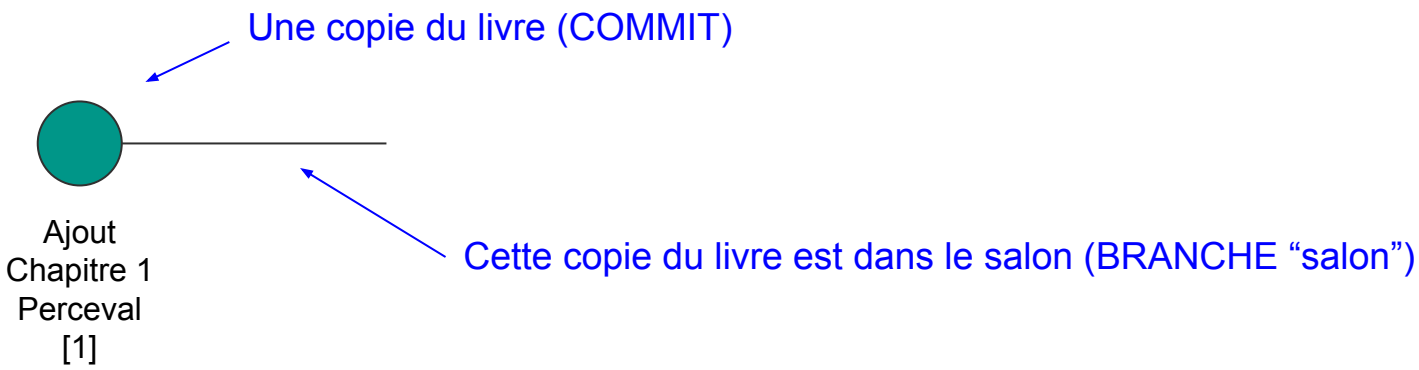
“C’est quoi ce git machin chose ?”

GIT

7.9.20

Perceval démarre l’écriture d’un livre dans son salon.

Il a terminé le chapitre 1.

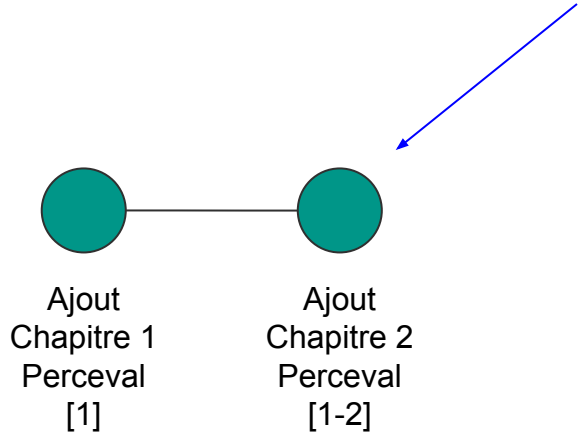


GIT

7.9.20

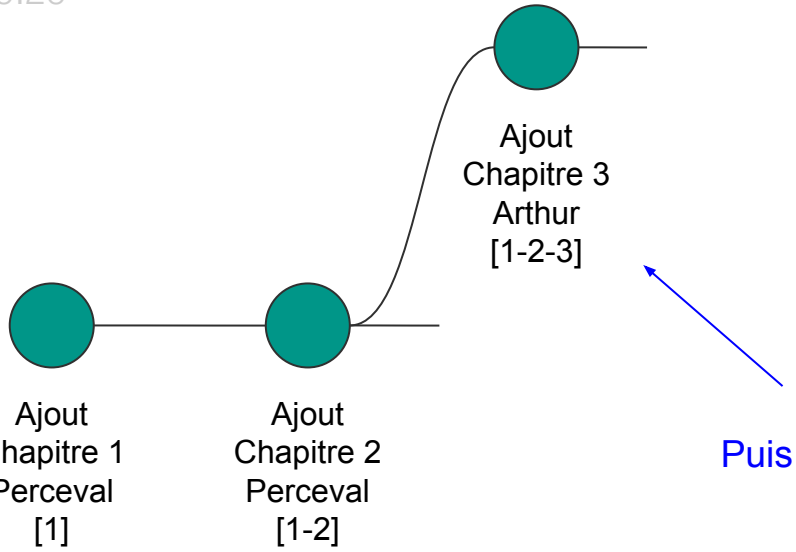
Perceval a ajouté un chapitre au livre dans le salon.

Cette nouvelle copie (COMMIT) contient maintenant les chapitre 1 et 2.



GIT

7.9.20

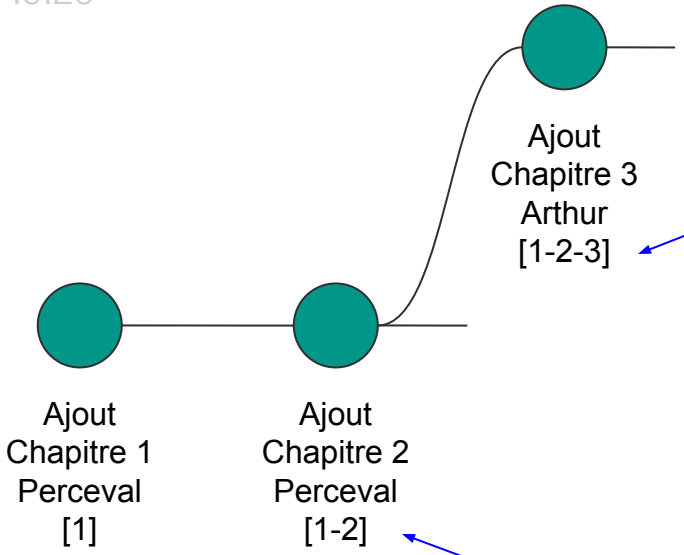


Arthur est allé dans la chambre avec une copie du livre du salon (NOUVELLE BRANCHE).

Puis il a écrit le chapitre 3 (COMMIT).

GIT

7.9.20

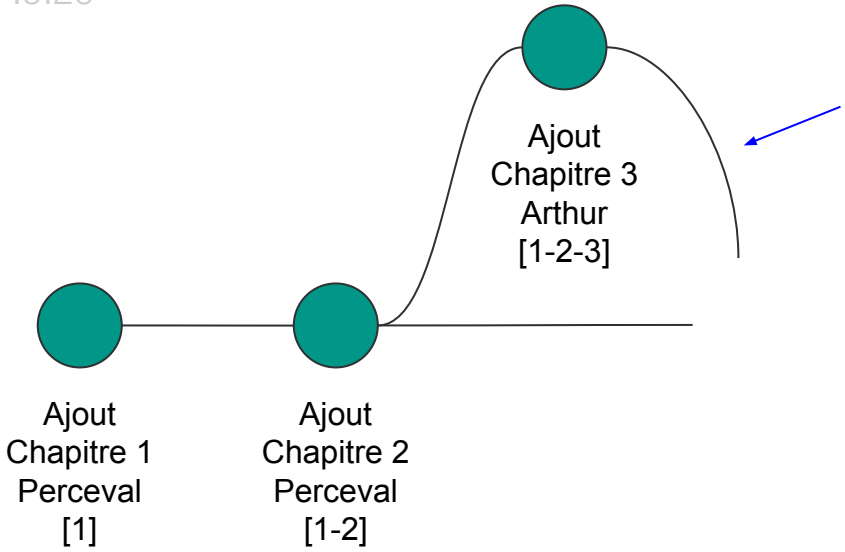


Dans la chambre, la copie contient le chapitre 1, 2 et 3.

Dans le salon, la copie ne contient que le chapitre 1 et 2.

GIT

7.9.20

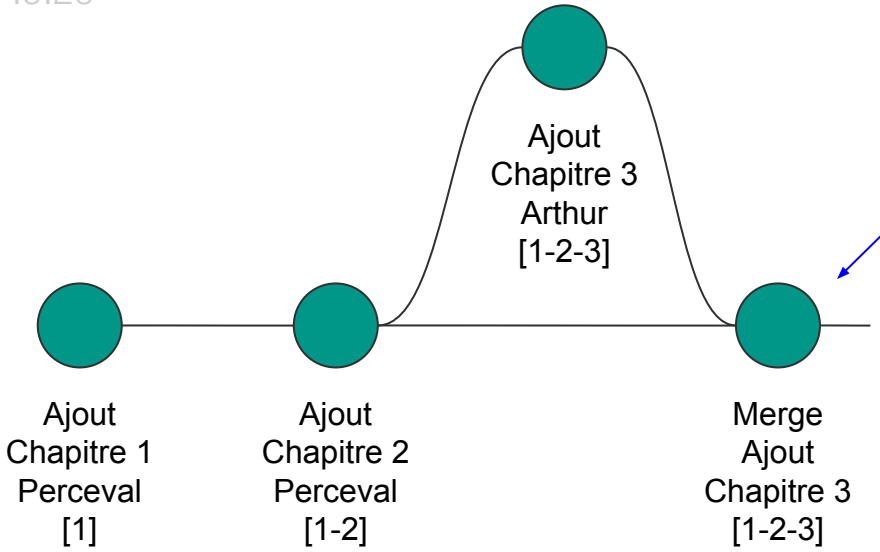


Arthur apporte sa copie à Perceval dans le salon.

Il lui demande de relire les modifications qu'il a apporté. (REVUE DE CODE / MERGE REQUEST)

GIT

7.9.20

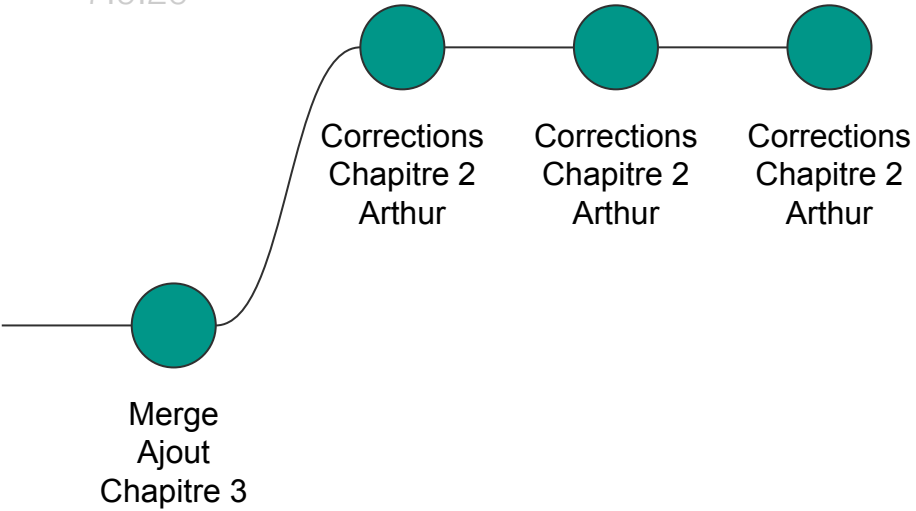


Perceval est d'accord, il accepte les modifications pour les mettre dans le livre du salon (MERGE).

Le livre du salon contient les 3 chapitres.

GIT

7.9.20

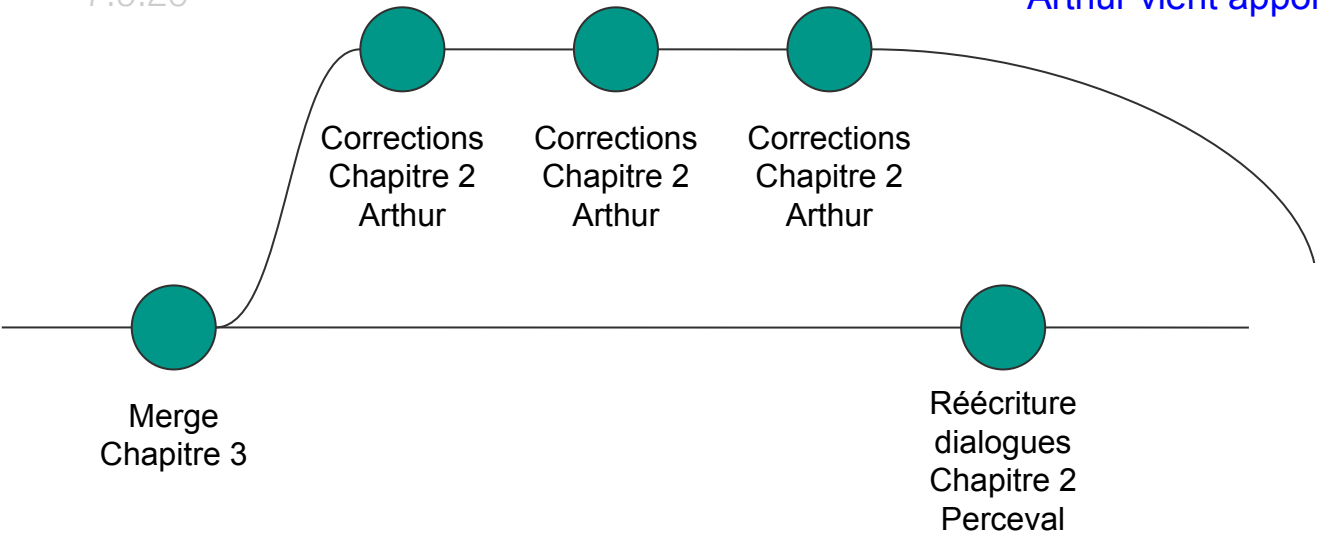


Arthur repart dans la chambre avec une nouvelle copie du livre. Il va corriger de nombreuses fautes d'orthographe du chapitre 2.

GIT

7.9.20

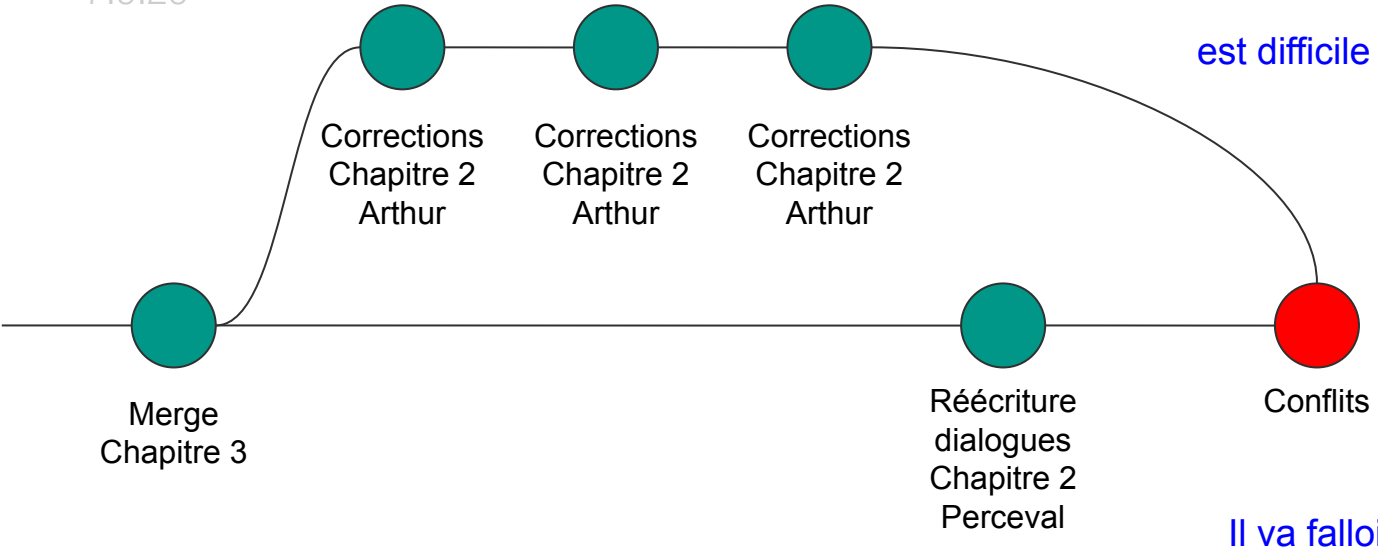
Arthur vient apporter ses corrections dans le salon.



Dans le salon, Perceval a aussi travaillé sur le chapitre 2, il a réécrit certains dialogues.

GIT

7.9.20



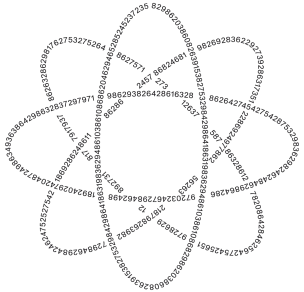
Réunir les changements dans le salon est difficile (CONFLITS DE MERGE).

Il va falloir résoudre les conflits en gardant, jetant ou modifiant les changements d'Arthur avant de réunir.

GIT

7.9.20

- Un commit est une copie à un instant T
- Les branches sont des espaces indépendants, avec copies
- Merger, c’est réunir des travaux fait séparément
    - Des conflits arrivent quand il y a des changements simultanés sur la même zone
    - Plus longtemps on travaille sur une branche, plus on augmente les problèmes



COLLABORER SUR LE CODE

3.15.12.12.1.2.15.18.5.18 19.21.18 12.5 3.15.4.5

“J’en ai marre de devoir refaire plusieurs fois mon code après des aller-retours sur la code review.”

“Les développeurs se plaignent que les revues de codes sont longues à faire et pas intéressantes.”

“On attends longtemps avant qu’un autre développeur fasse la revue de code.”

REVUE DE CODE

18.5.22.21.5 4.5 3.15.4.5

- Asynchrone (changements de contexte)
- Feedback tardif
- Goulot d’étranglement
- Bâclée (quand les problèmes se cumulent)

Pull/Merge Request, davantage de sens dans l’open-source.

PAIR PROGRAMMING

16.1.9.18 16.18.15.7.18.1.13.13.9.14.7

- Réelle collaboration de deux personnes sur un problème
- Feedback immédiat
- Forte transmission de connaissances
    - Techniques et fonctionnelles

Et sans diviser l’efficacité par 2 :)

MOB PROGRAMMING

13.15.2 16.18.15.7.18.1.13.13.9.14.7

- Réelle collaboration de tout un groupe sur un problème
- Réflexion commune qui mène à une meilleure conception
- Très forte transmission de connaissances
- Collective Ownership

COLLABORER SUR LE CODE

3.15.12.12.1.2.15.18.5.18 19.21.18 12.5 3.15.4.5

“Le dev est terminé localement mais ça met beaucoup de temps à être livré car on a plein de conflits.”

“La fonctionnalité marche sur ma branche mais maintenant que c’est mergé c’est cassé.”

INTÉGRATION CONTINUE

9.14.20.5.7.18.1.20.9.15.14 3.15.14.20.9.14.21.5

- C’est une attitude !
    - Rassembler très souvent nos changements sur la branche principale
- De plus petites étapes
- Davantage de tests automatisés
- Diminution des conflits

PAIR/MOB PROGRAMMING

16.1.9.18 13.15.2 16.18.15.7.18.1.13.13.9.14.7

- Travail à plusieurs
- Moins d’en cours
- Communication accrue et rapide

=> Diminution des conflits

ÉCRIRE LE CODE

5.3.18.9.18.5 12.5 3.15.4.5

“On a fait évolué l’écran recherche, et on a vu plus tard en prod que la logique métier du compte est complètement KO.”

“On planifie de passer une après midi sur l’écriture de tests, on fait comment pour le prendre en compte dans la vélocité ?”

“J’ai terminé ma story, est-ce que j’ai le temps pour écrire des tests ?”

TESTS UNITAIRES

20.5.19.20.19 21.14.9.20.1.9.18.5.19

- Test automatisé d’un scénario fonctionnel
- S’assurer de bien retranscrire le métier, le comportement
- Filet pour les régressions, feedback tôt
- Essentiel au refactoring

TEST-DRIVEN DEVELOPMENT

20.5.19.20 4.18.9.22.5.14 4.5.22.5.12.15.16.13.5.14.20

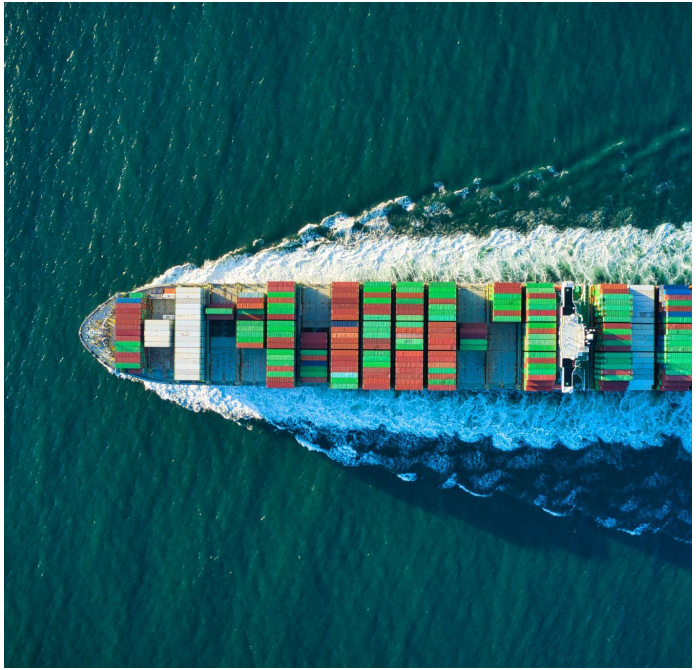
- Être guidé par les tests et le métier
- Faire émerger la solution
- Boucle de feedback très rapide

Ça fait partie du job.

Et ce n’est pas plus long.

LIVRER

12.9.22.18.5.18



“C’est douloureux de livrer, c’est long, il y a un tas de process et d’actions à faire…”

“On va s’assurer d’avoir suffisamment de choses dans la release pour livrer. C’est long alors il faut que ça vaille le coup.”

“On ne livre pas le vendredi” / “On ne peut rien modifier en août et fin décembre. On a un code freeze pendant les périodes avec beaucoup de vacances.”

AUTOMATISATION

1.21.20.15.13.1.20.9.19.1.20.9.15.14

- Automatiser des tâches répétitives
- Gain de temps et de cohérence
- Moins d’erreurs humaines

LIVRAISON CONTINUE

12.9.22.18.1.9.19.15.14 3.15.14.20.9.14.21.5

- Chaque changement sur la branche principale est livré (staging/prod*)
- Reste un simple bouton pour déployer aux utilisateurs
- Qui peut le plus, peut le moins
- Petits incréments
- Le code est toujours dans un état déployable

DÉPLOIEMENT CONTINU

4.5.16.12.15.9.5.13.5.14.20 3.15.14.20.9.14.21

- Chaque changement sur la branche principale va automatiquement en production
- Aucune intervention humaine
- On est dans un cercle vertueux de pratiques

AMÉLIORER UN EXISTANT

1.13.5.12.9.15.18.5.18 21.14 5.24.9.19.20.1.14.20

“On a des problèmes de qualité dont on n’a jamais le temps de s’occuper.”

“Mon équipe veut négocier un sprint ou deux de refactoring mais je ne sais pas quelle valeur ça va apporter.”

“J’ai un développeur qui fait du refactoring depuis 3 mois, c’est pas encore fini mais il avance*.”

** Spoiler : ce sera probablement abandonné avant la fin :)*

REFACTORING

18.5.6.1.3.20.15.18.9.14.7

- Changer le code sans casser le comportement observable
- Série de petites étapes
- Amélioration continue et quotidienne

REFACTORING

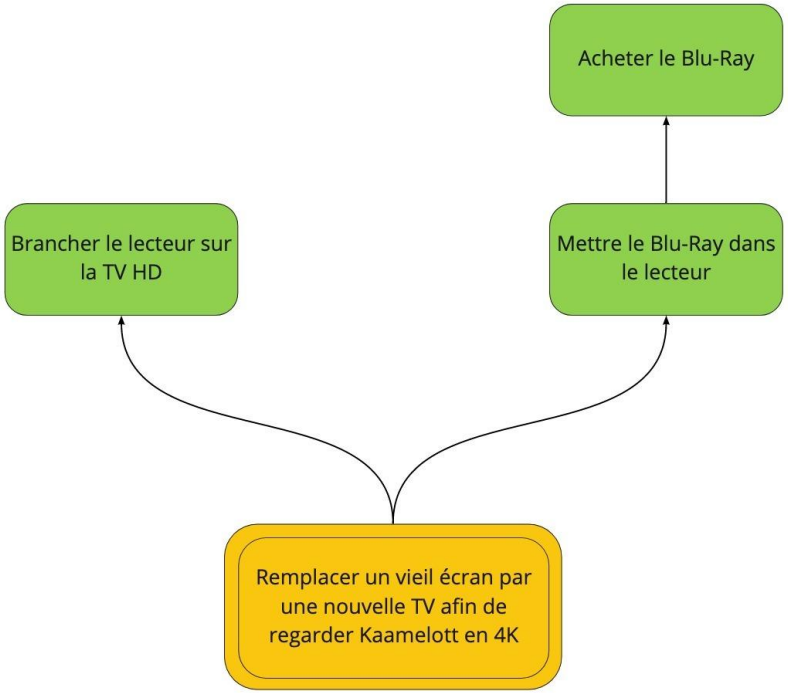
18.5.6.1.3.20.15.18.9.14.7

- Règle du Boy Scout
    - Rendre le code plus propre à chaque occasion
- Théorie de la vitre brisée
    - Remédier avant que ça ne s’accumule

MÉTHODE MIKADO

13.5.20.8.15.4.5 13.9.11.1.4.15

- Atteindre un objectif par petites étapes
- Construire un chemin via des expérimentations
- Dessiner un graph pour visualiser



Ça fait partie du job.

Et il faut se former.

BREF

2.18.5.6

Example Mapping
Pair Programming
Mob Programming
Tests Unitaires
Test-Driven Development
Intégration Continue
Automatisation
Livraison Continue
Déploiement Continu
Refactoring
…

Des pratiques pour améliorer le **quotidien** de l'équipe et devenir meilleur dans la **technique** et dans le **business**.

Blog - https://jordanchapuy.com

Twitter - @Yupjoo

MERCI

Questions, cris du cœur, …